JAVA-8 Stream and Lambda

1. PROBLEM STATEMENT

You manage a library and need to implement a digital catalog system for books. Utilize Java 8 stream concepts to achieve the following functionalities:

Requirement:

1. **Retrieve books grouped by genre.**
   - Implement a method `retrieveBooksGroupedByGenre(Stream<Book> books)` that takes a stream of `Book` objects and groups them by their genre. Return a map where each key is a genre and the value is a list of books in that genre.

2. **Get authors and prices for a specified book title.**
   - Implement a method `getAuthorsAndPricesByTitle(Stream<Book> books, String title)` that takes a stream of `Book` objects and a `title` as input parameters. Return a map where the keys are the authors of books with the specified title, and the values are the prices of those books.

3. **Generate summary reports for available books.**
   - Implement a method `generateSummaryReport(List<Book> books)` that takes a list of `Book` objects and returns a stream of strings summarizing the books. Calculate and include:
     - Total number of books as "Total books: <total number of books>"
     - Total price of all books as "Total price: <total price of all books>"
     - Total quantity (number of copies) of all books as "Total quantity: <total quantity of all books>"

Component Specification:

Book (POJO class)

- Attributes:
  - String title
  - String genre
  - String author
  - int quantity
  - double price
- Methods:
  - Getters and setters
  - Constructors (default, and parameterized with all attributes)

BookUtility (Utility class)

- Methods:
  - `public Map<String, List<Book>> retrieveBooksGroupedByGenre(Stream<Book> books)`
  - `public Map<String, Double> getAuthorsAndPricesByTitle(Stream<Book> books, String title)`
  - `public Stream<String> generateSummaryReport(List<Book> books)`

Conditions:

- Title comparison is case-sensitive.
- Ensure object-oriented principles and method signatures as specified.
- Assume all inputs are valid and handle cases where no books match a given title appropriately.

Sample Input / Output:
```
Enter the number of books you want to add:
3

Enter the details of the books:
Java Programming:Computer Science:John Doe:10:45.99
Python Basics:Computer Science:Jane Smith:8:39.95
Data Structures:Computer Science:John Doe:5:55.50

Books grouped by genre:
Computer Science
Java Programming 10 John Doe 45.99
Python Basics 8 Jane Smith 39.95
Data Structures 5 John Doe 55.5

Enter book title to get authors and prices:
Python Basics

Authors and prices of Python Basics:
Jane Smith 39.95

Summary Report:
Total books: 3
Total price: 141.44
Total quantity: 23
```
==

2. PROBLEM STATEMENT

You manage a movie rental service and need to implement a digital catalog system for movies. Use Java 8 stream concepts to achieve the following functionalities:

****Requirement:****

1. **Retrieve movies grouped by genre.**
   - Implement a method `retrieveMoviesGroupedByGenre(Stream<Movie> movies)` that takes a stream of `Movie` objects and groups them by their genre. Return a map where each key is a genre and the value is a list of movies in that genre.

2. **Get directors and rental prices for a specified movie title.**
   - Implement a method `getDirectorsAndPricesByTitle(Stream<Movie> movies, String title)` that takes a stream of `Movie` objects and a `title` as input parameters. Return a map where the keys are the directors of movies with the specified title, and the values are the rental prices of those movies.

3. **Calculate late fees for overdue rentals.**
   - Implement a method `calculateLateFees(List<Rental> rentals, int daysLate)` that takes a list of `Rental` objects and the number of days late as parameters. Calculate and return the total late fees incurred based on a fixed rate per day late.

4. **Generate summary reports for available movies.**
   - Implement a method `generateSummaryReport(List<Movie> movies)` that takes a list of `Movie` objects and returns a stream of strings summarizing the movies. Calculate and include:
     - Total number of movies as "Total movies: <total number of movies>"
     - Total rental revenue as "Total revenue: <total revenue from all rentals>"
     - Average rental price as "Average price: <average rental price of all movies>"

5. **Search for movies by director.**
   - Implement a method `searchMoviesByDirector(Stream<Movie> movies, String directorName)` that takes a stream of `Movie` objects and a director's name as input. Return a list of movies directed by the specified director.

6. **Sort movies by rental price.**
   - Implement a method `sortMoviesByPrice(Stream<Movie> movies, boolean ascendingOrder)` that takes a stream of `Movie` objects and a boolean flag indicating ascending or descending order. Return a sorted list of movies based on their rental prices.

Component Specification:

Movie (POJO class)

- Attributes:
  - String title
  - String genre
  - String director
  - int quantityAvailable
  - double rentalPrice
- Methods:
  - Getters and setters
  - Constructors (default, and parameterized with all attributes)

Rental (POJO class)

- Attributes:
  - String customerName
  - String movieTitle
  - LocalDate rentalDate
  - int daysRented
- Methods:
  - Getters and setters
  - Constructors (default, and parameterized with all attributes)

MovieCatalogUtility (Utility class)

- Methods:
  - `public Map<String, List<Movie>> retrieveMoviesGroupedByGenre(Stream<Movie> movies)`
  - `public Map<String, Double> getDirectorsAndPricesByTitle(Stream<Movie> movies, String title)`
  - `public double calculateLateFees(List<Rental> rentals, int daysLate)`
  - `public Stream<String> generateSummaryReport(List<Movie> movies)`
  - `public List<Movie> searchMoviesByDirector(Stream<Movie> movies, String directorName)`
  - `public List<Movie> sortMoviesByPrice(Stream<Movie> movies, boolean ascendingOrder)`

Conditions:

- Title comparison is case-sensitive.
- Assume all inputs are valid and handle cases where no movies match a given title or director appropriately.
- Late fees are calculated at a fixed rate per day late, specified in the method implementation.

- Sorting functionality should be implemented using Java stream API methods.

Sample Input / Output:

```
Enter the number of movies you want to add:
5

Enter the details of the movies:
Inception:Sci-Fi:Christopher Nolan:10:3.99
The Matrix:Sci-Fi:Lana Wachowski:8:2.99
The Dark Knight:Action:Christopher Nolan:5:4.99
Joker:Drama:Todd Phillips:6:3.49
Interstellar:Sci-Fi:Christopher Nolan:7:3.79

Movies grouped by genre:
Sci-Fi
Inception 10 Christopher Nolan 3.99
The Matrix 8 Lana Wachowski 2.99
Interstellar 7 Christopher Nolan 3.79

Action
The Dark Knight 5 Christopher Nolan 4.99

Drama
Joker 6 Todd Phillips 3.49

Enter movie title to get directors and prices:
The Matrix

Directors and prices of The Matrix:
Lana Wachowski 2.99

Movies directed by Christopher Nolan:
Inception Sci-Fi 10 3.99
The Dark Knight Action 5 4.99
Interstellar Sci-Fi 7 3.79

Summary Report:
Total movies: 5
Total revenue: 74.34
Average price: 3.97

Movies sorted by rental price (ascending):
```

The Matrix Sci-Fi 8 2.99
Interstellar Sci-Fi 7 3.79
Inception Sci-Fi 10 3.99
Joker Drama 6 3.49
The Dark Knight Action 5 4.99
```
===================================================================

## 3. PROBLEM STATEMENT

As you developed an application for managing employee records using Java streams, you aimed to implement functionalities to filter and display employees based on specific criteria.

**Requirements:**

**1. **Get employees by department:**** Filter employees based on the specified department and display details.
**2. **Get employees by location:**** Filter employees based on the specified location and display details.
**3. **Get employees by salary range:**** Filter employees based on the specified salary range and display details.

**Component Specification: Employee (POJO class)**
Employee
String name
String department
String location
double salary
int experience
Getters and setters, no argument, and four argument constructors ```

**Component Specification: EmployeeUtility**

**EmployeeUtility**

**public Stream<Employee> getEmployeesByDepartment(Stream<Employee> employeeStream, String department)**

The method takes a stream of Employee objects and department as input parameters, and it should return a stream of Employee objects. This method achieves this by filtering the employees from the provided employeeStream based on the given department.
```

Condition:
department is case-insensitive.

EmployeeUtility public List<Employee> getEmployeesByLocation(Stream<Employee> employeeStream, String location)

The method takes a stream of Employee objects and location as input parameters, and it should return a list of Employee objects. This method achieves this by filtering the employees from the provided employeeStream based on the given location.

Condition:
location is case-insensitive.

EmployeeUtility public Stream<Employee> getEmployeesInSalaryRange(List<Employee> employeeList, double minSalary, double maxSalary)

The method takes a list of Employee objects, minimum salary, and maximum salary as input parameters, and it should return a stream containing Employee objects. This method achieves this by filtering the list of employees based on whether their salary falls within the specified range.

Condition:
minSalary should be less than or equal to maxSalary.

The main method in the UserInterface class allows the user to input the total number of employees and their details, which are then stored in a list of Employee objects.

Implement the following functionalities:

1. Get the department from the user. Invoke the `getEmployeesByDepartment` method to filter employees by department. If employees are available for the given department, print their details in the format: `<name>:<department>:<location>:<salary>`. Otherwise, print "No employees found for the given department".

2. Get the location from the user. Invoke the `getEmployeesByLocation` method to filter employees by location. If employees are available for the given location, print their details in the format: `<name>:<department>:<location>:<salary>`. Otherwise, print "No employees found for the given location".

3. Get the salary range from the user. Invoke the `getEmployeesInSalaryRange` method to filter employees by salary range. If employees are available within the

specified salary range, print their details in the format: `<name>:<department>:<location>:<salary>`. Otherwise, print "No employees found within the given salary range".

Sample Input / Output:

```

Enter the total number of employees needed to add in the list

3

Enter the employee details

John Doe:Finance:New York:75000
Jane Smith:HR:Los Angeles:60000
Michael Johnson:Engineering:San Francisco:90000

Enter the department

Finance

**John Doe:Finance:New York:75000**

Enter the location

Los Angeles

**Jane Smith:HR:Los Angeles:60000**

Enter the salary range (min max)

60000 80000

**John Doe:Finance:New York:75000**
**Jane Smith:HR:Los Angeles:60000**

This problem statement includes a sample interaction where the user inputs employee details, filters employees by department, location, and salary range, and displays the results accordingly.

==============================================================
```

4. PROBLEM STATEMENT

The Digital Library Consortium (DLC) wants to enhance their system for managing digital book records using Java Streams, making it more efficient to access information such as book ID, title, author, genre, price, and availability status. This new system aims to streamline data organization and retrieval, ensuring quick access to book details.

Create a Java application that uses streams to perform the following tasks:

****Requirements:****

1. Retrieve book details based on book ID.
2. Retrieve book details based on genre.
3. Get the list of book IDs within a specified price range.

****Component Specification: Book (POJO Class)****

- Attributes:
  - String bookId
  - String title
  - String author
  - String genre
  - double price
  - boolean available

- Methods:
  - Getters and setters
  - Constructors (default and parameterized)

****Component Specification: LibraryUtil (Utility class)****

- Methods:
  - `public Book retrieveBookDetailsById(Stream<Book> bookStream, String bookId)`
    - Retrieves book details from the provided stream based on the given book ID.
    - Returns a Book object if found, otherwise null.

  - `public List<Book> retrieveBooksByGenre(Stream<Book> bookStream, String genre)`
    - Retrieves a list of Book objects from the stream based on the specified genre.
    - Returns an empty list if no books are found for the given genre.

  - `public List<String> retrieveBookIdsInPriceRange(Stream<Book> bookStream, double minPrice, double maxPrice)`

    - Retrieves a list of book IDs from the stream where the book price falls within the specified range (inclusive).
    - Returns an empty list if no books are found within the price range.

UserInterface Class (Main class)

The main method in the `UserInterface` class will:
- Prompt the user to input the number of books to be added.
- Accept details for each book (book ID, title, author, genre, price, availability).
- Store these books in a list.
- Allow users to input:
  - A book ID to retrieve specific book details.
  - A genre to retrieve all books of that genre.
  - A price range to retrieve book IDs within that range.

The output should:
- Display the retrieved book details in the specified format or indicate if no results are found based on user input.

Sample Input / Output:

```
Enter the number of books to be added: 4

Enter book details:
Book ID, Title, Author, Genre, Price, Availability
B001, The Great Gatsby, F. Scott Fitzgerald, Fiction, 12.99, true
B002, Algorithms Unlocked, Thomas Cormen, Computer Science, 34.50, true
B003, Java Programming, John Doe, Computer Science, 45.00, true
B004, Pride and Prejudice, Jane Austen, Fiction, 9.99, true

Enter the Book ID to retrieve details: B002

B002 Algorithms Unlocked Thomas Cormen Computer Science 34.5 true

Enter the Genre to retrieve details: Fiction

B001 The Great Gatsby F. Scott Fitzgerald Fiction 12.99 true
B004 Pride and Prejudice Jane Austen Fiction 9.99 true

Enter the minimum and maximum price range to show book IDs: 20.00 40.00

Book IDs within the price range are:
B002
```